WARWICK
THE UNIVERSITY OF WARWICK

Original citation:
Lazic, Ranko and Murawski, Andrzej S. (2016) Contextual approximation and higher-order procedures. In: 19th International Conference, FOSSACS 2016, Eindhoven, The Netherlands, 2-8 Apr 2016. Published in: Foundations of Software Science and Computation Structures :19th International Conference, FOSSACS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceeding, 9634 pp. 162-179.

Permanent WRAP URL:
http://wrap.warwick.ac.uk/78376

Copyright and reuse:
The Warwick Research Archive Portal (WRAP) makes this work by researchers of the University of Warwick available open access under the following conditions. Copyright © and all moral rights to the version of the paper presented here belong to the individual author(s) and/or other copyright owners. To the extent reasonable and practicable the material made available in WRAP has been checked for eligibility before being made available.

Copies of full items can be used for personal research or study, educational, or not-for profit purposes without prior permission or charge. Provided that the authors, title and full bibliographic details are credited, a hyperlink and/or URL is given for the original metadata page and the content is not changed in any way.

Publisher's statement:
"The final publication is available at Springer"
http://dx.doi.org/10.1007/978-3-662-49630-5_10
A note on versions:
The version presented here may differ from the published version or, version of record, if you wish to cite this item you are advised to consult the publisher's version. Please see the 'permanent WRAP url' above for details on accessing the published version and note that access may require a subscription.

For more information, please contact the WRAP Team at: wrap@warwick.ac.uk

warwick.ac.uk/lib-publications

Contextual approximation and higher-order procedures*

Ranko Lazić and Andrzej S. Murawski

DIMAP and Department of Computer Science, University of Warwick, UK

Abstract. We investigate the complexity of deciding contextual approximation (refinement) in finitary Idealized Algol, a prototypical language combining higher-order types with state. Earlier work in the area established the borderline between decidable and undecidable cases, and focussed on the complexity of deciding approximation between terms in normal form.
In contrast, in this paper we set out to quantify the impact of locally declared higher-order procedures on the complexity of establishing contextual approximation in the decidable cases. We show that the obvious decision procedure based on exhaustive β-reduction can be beaten. Further, by classifying redexes by levels, we give tight bounds on the complexity of contextual approximation for terms that may contain redexes up to level k, namely, $(k-1)$-EXPSPACE-completeness. Interestingly, the bound is obtained by selective β-reduction: redexes from level 3 onwards can be reduced without losing optimality, whereas redexes up to order 2 are handled by a dedicated decision procedure based on game semantics and a variant of pushdown automata.

1 Introduction

Contextual approximation (refinement) is a fundamental notion in programming language theory, facilitating arguments about program correctness [17] as well as supporting formal program development [6]. Intuitively, a term M_1 is said to *contextually approximate* another term M_2, if substituting M_1 for M_2 in any context will not lead to new observable behaviours. Being based on universal quantification over contexts, contextual approximation is difficult to establish directly. In this paper, we consider the problem of contextual approximation in a higher-order setting with state. Contextual reasoning at higher-order types is a recognised challenge and a variety of techniques have been proposed to address it, such as Kripke logical relations [3] or game models [2]. In this work, we aim to understand the impact of locally defined higher-order procedures on the complexity of establishing contextual approximation. Naturally, one would expect the complexity to grow in the presence of procedures and it should grow as the type-theoretic order increases. We shall quantify that impact by providing tight complexity bounds for contextual approximation in our higher-order framework. The results demonstrate that, from the complexity-theoretic point of view, it is safe to inline procedures only down to a certain level. Below that level, however, it is possible to exploit compositionality to arrive at better bounds than those implied by full inlining.

The vehicle for our study is Idealized Algol [15, 1], the protypical language for investigating the combination of local state with higher-order procedures. In order to

* Research supported by EPSRC (EP/J019577/1, EP/M011801/1).

avoid obviously undecidable cases, we restrict ourselves to its finitary variant IA_{f}, featuring finite base types and no recursion (looping is allowed, though). Both semantic and syntactic methods were used to reason about Idealized Algol [1, 14] in the past. In particular, on the semantic front, there exists a game model that captures contextual approximation (in the sense of inequational full abstraction) via complete-play inclusion. Earlier work in the area [8, 11, 13] used this correspondence to map out the borderline between decidable and undecidable cases within IA_{f}. The classification is based on type-theoretic order: a term is of order i if its type is of order at most i and all free variables have order less than i. We write IA_i for the set of IA_{f}-terms of order i. It turns out that contextual approximation is decidable for terms of all orders up to 3, but undecidable from order 4 onwards. The work on decidability has also established accurate complexity bounds for reasoning about contextual approximation between terms in β-normal form as well as terms with the simplest possible β-redexes, in which arguments can only be of base type. For order-3 terms, the problem can be shown EXPTIME-complete [13], while at orders $0, 1, 2$ it is PSPACE-complete [12]. In this paper, we present a finer analysis of the decidable cases and consider arbitrary β-redexes. In particular, functions can be used as arguments, which corresponds to the inlining of procedures.

We evaluate the impact of redexes by introducing a notion of their level: the level of a β-redex $(\lambda x.M)N$ will be the order of the type of $\lambda x.M$. Accordingly, we can split IA_i into sublanguages IA_i^k, in which terms can contain redexes of level up to k. IA_i^0 is then the normal-form case and IA_i^1 is the case of base-type arguments. Obviously, the problem of contextually approximating IA_i^k ($i \leq 3, k \geq 2$) terms can be solved by β-reduction (and an appeal to the results for IA_i^0), but this is known to result in a k-fold exponential blow-up, thus implying a $(k+1)$-EXPTIME upper bound for IA_3^k. This bound turns out to be suboptimal. One could lower it by reducing to IA_i^1 instead, which would shave off a single exponential, but this is still insufficient to arrive at the optimal bound. It turns out, however, that reducing IA_3^k terms to IA_3^2 (all redexes up to order 3 are eliminated) does not lead to a loss of optimality. To work out the accurate bound for the IA_3^2 case, one cannot apply further β-reductions, though. Instead we devise a dedicated procedure based on game semantics and pushdown automata. More specifically, we introduce a variant of visibly pushdown automata [4] with ϵ-transitions and show how to translate IA_3^2 into such automata so that the accepted languages are faithful representations of the term's game semantics [1]. The translation can be performed in exponential time and, crucially, the automata correspoding to IA_3^2-terms satisfy a boundedness property: the stack symbols pushed on the stack during ϵ-moves can only form contiguous segments of exponential length with respect to the term size. This allows us to solve the corresponding inclusion problem in exponential space with respect to the original term size. Consequently, we can show that IA_3^2 contextual approximation is in EXPSPACE.

The above result then implies that program approximation of IA_3^k-terms is in $(k-1)$-EXPSPACE. Furthermore, we can prove matching lower bounds for IA_1^k. The table below summarises the complexity results. The results for $k \geq 2$ are new.

	$k=0$	$k=1$	$k \geq 2$
IA_1^k	PSPACE-complete [12]	PSPACE-complete [12]	$(k-1)$-EXPSPACE-complete
IA_2^k	PSPACE-complete [12]	PSPACE-complete [12]	$(k-1)$-EXPSPACE-complete
IA_3^k	EXPTIME-complete [13]	EXPTIME-complete [13]	$(k-1)$-EXPSPACE-complete

2 Idealized Algol

We consider a finitary version IA_f of Idealized Algol with active expressions [1]. Its types are generated by the following grammar.

$$\theta ::= \beta \mid \theta \to \theta \qquad \beta ::= \mathsf{com} \mid \mathsf{exp} \mid \mathsf{var}$$

IA_f can be viewed as a simply-typed λ-calculus over the base types $\mathsf{com}, \mathsf{exp}, \mathsf{var}$ (of commands, expressions and variables respectively) augmented with the constants listed below

$$\begin{array}{c} \mathbf{skip} : \mathsf{com} \qquad i : \mathsf{exp} \quad (0 \le i \le max) \qquad \mathbf{succ} : \mathsf{exp} \to \mathsf{exp} \qquad \mathbf{pred} : \mathsf{exp} \to \mathsf{exp} \\ \mathbf{ifzero}_\beta : \mathsf{exp} \to \beta \to \beta \to \beta \qquad \mathbf{seq}_\beta : \mathsf{com} \to \beta \to \beta \qquad \mathbf{deref} : \mathsf{var} \to \mathsf{exp} \\ \mathbf{assign} : \mathsf{var} \to \mathsf{exp} \to \mathsf{com} \qquad \mathbf{cell}_\beta : (\mathsf{var} \to \beta) \to \beta \\ \mathbf{while} : \mathsf{exp} \to \mathsf{com} \to \mathsf{com} \qquad \mathbf{mkvar} : (\mathsf{exp} \to \mathsf{com}) \to \mathsf{exp} \to \mathsf{var} \end{array}$$

where β ranges over base types and $\mathsf{exp} = \{ 0, \cdots, max \}$. Other IA_f-terms are formed using λ-abstraction and application

$$\frac{\Gamma \vdash M : \theta \to \theta' \quad \Gamma \vdash N : \theta}{\Gamma \vdash MN : \theta'} \qquad \frac{\Gamma, x : \theta \vdash M : \theta'}{\Gamma \vdash \lambda x^\theta.M : \theta \to \theta'}$$

using the obvious rules for constants and free identifiers. Each of the constants corresponds to a different programming feature. For instance, the sequential composition of M and N (typically denoted by $M; N$) is expressed as $\mathbf{seq}_\beta MN$, assignment of N to M ($M := N$) is represented by $\mathbf{assign}MN$ and $\mathbf{cell}_\beta(\lambda x.M)$ amounts to creating a local variable x visible in M (**new** x **in** M). $\mathbf{mkvar}$ is the so-called bad-variable constructor that makes it possible to construct terms of type var with prescribed read- and write-methods. $\mathbf{while}MN$ corresponds to **while** M **do** N. We shall write Ω_β for the divergent constant that can be defined using **while** 1 **do** $\mathbf{skip}$.

The operational semantics of IA_f, based on call-by-name evaluation, can be found in [1]; we will write $M \Downarrow$ if M reduces to $\mathbf{skip}$. We study the induced contextual approximation.

Definition 1. *We say that* $\Gamma \vdash M_1 : \theta$ contextually approximates $\Gamma \vdash M_2 : \theta$ *if, for any context* $C[-]$ *such that* $C[M_1], C[M_2]$ *are closed terms of type* com*, we have* $C[M_1]\Downarrow$ *implies* $C[M_2]\Downarrow$*. We then write* $\Gamma \vdash M_1 \sqsubseteq_\sim M_2$*.*

Even though the base types are finite, IA_f contextual approximation is not decidable [11]. To obtain decidability one has to restrict the order of types, defined by:

$$\mathsf{ord}(\beta) = 0 \qquad \mathsf{ord}(\theta \to \theta') = \max(\mathsf{ord}(\theta) + 1, \mathsf{ord}(\theta')).$$

Definition 2. *Let* $i \ge 0$*. The fragment* IA_i *of* IA_f *consists of* IA_f*-terms* $x_1 : \theta_1, \cdots, x_n : \theta_n \vdash M : \theta$ *such that* $\mathsf{ord}(\theta_j) < i$ *for any* $j = 1, \cdots, n$ *and* $\mathsf{ord}(\theta) \le i$*.*

Contextual approximation is known to be decidable for IA_1, IA_2 and IA_3 [13], but it is undecidable for IA_4 [11].

Definition 3. – *The* level *of a* β*-redex* $(\lambda x^\theta.M)N$ *is the order of the type of* $\lambda x^\theta.M$*.*

$$\begin{array}{ll} M_{A\times B} = M_A + M_B & M_{A\Rightarrow B} = M_A + M_B \\ \lambda_{A\times B} = [\lambda_A, \lambda_B] & \lambda_{A\Rightarrow B} = [\overline{\lambda}_A, \lambda_B] \\ \vdash_{A\times B} = \vdash_A \ + \vdash_B & \vdash_{A\Rightarrow B} = \vdash_B \ + (I_B \times I_A) \ + \ (\vdash_A \cap (M_A \times M_A)) \end{array}$$

$\overline{\lambda}_A$ reverses the ownership of moves in A while preserving their kind.

Fig. 1. Constructions on arenas

- *A term has* degree k *if all redexes inside it have level at most* k*.*
- IA_i^k *is the subset of* IA_i *consisting of terms whose degree is at most* k*.*

β-reduction can be used to reduce the degree of a term by one at an exponential cost.

Lemma 1. *Let* $d \geq 1$*. A* λ*-term* M *of degree* d *can be reduced to a term* M' *of degree* $d-1$ *with a singly-exponential blow-up in size.*

3 Games

Game semantics views computation as an exchange of moves between two players, called O and P. It interprets terms as strategies for P in an abstract game derived from the underlying types. The moves available to players as well as the rules of the game are specified by an arena.

Definition 4. *An* arena *is a triple* $A = \langle\, M_A, \lambda_A, \vdash_A \,\rangle$*, where*

- M_A *is a set of* moves*;*
- $\lambda_A : M_A \to \{\, O, P \,\} \times \{\, Q, A \,\}$ *is a function indicating to which player (*O *or* P*) a move belongs and of what kind it is (question or answer);*
- $\vdash_A \subseteq (M_A + \{\, \star \,\}) \times M_A$ *is the so-called* enabling *relation, which must satisfy the following conditions:*
  - *If* $\star$ *enables a move then it is an O-question without any other enabler. A move like this is called* initial *and we shall write* I_A *for the set containing all initial moves of* A*.*
  - *If one move enables another then the former must be a question and the two moves must belong to different players.*

Arenas used to interpret the base types of IA_f are shown in Figure 2: the moves at the bottom are answer-moves. Product and function-space arenas can be constructed as shown in Figure 1. Given an IA_f-type θ, we shall write $[\![\theta]\!]$ for the corresponding arena obtained compositionally from A_{com}, A_{exp} and A_{var} using the $\Rightarrow$ construction. Given arenas, we can play games based on a special kind of sequences of moves. A *justified sequence* s in an arena A is a sequence of moves in which every move $m \notin I_A$ must have a pointer to an earlier move n in s such that $n \vdash_A m$. n is then said to be the *justifier* of m. It follows that every justified sequence must begin with an O-question.

Given a justified sequence s, its O-view $\llcorner s \lrcorner$ and P-view $\ulcorner s \urcorner$ are defined as follows, where o and p stand for an O-move and a P-move respectively:

$$\begin{array}{llll} \llcorner \epsilon \lrcorner = \epsilon & \llcorner so \lrcorner = \llcorner s \lrcorner o & & \llcorner s\, o\overset{\frown}{\ t\ } p \lrcorner = \llcorner s \lrcorner o\overset{\frown}{\ } p \\ \ulcorner \epsilon \urcorner = \epsilon & \ulcorner so \urcorner = o \quad \text{(if } o \text{ is initial)} & \ulcorner sp \urcorner = \ulcorner s \urcorner p & \ulcorner s\, p\overset{\frown}{\ t\ } o \urcorner = \ulcorner s \urcorner p \frown o. \end{array}$$

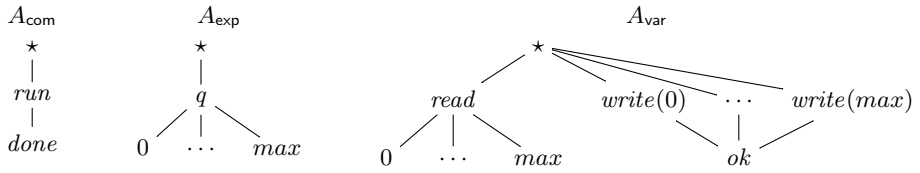

Fig. 2. Arenas for base types

A justified sequence s satisfies *visibility* condition if, in any prefix tm of s, if m is a non-initial O-move then its justifier occurs in $\llcorner t \lrcorner$ and if m is a P-move then its justifier is in $\ulcorner t \urcorner$. A justified sequence satisfies the *bracketing* condition if any answer-move is justified by the latest unanswered question that precedes it.

Definition 5. *A justified sequence is a* play *iff O- and P-moves alternate and it satisfies bracketing and visibility. We write P_A for the set of plays in an arena A. A play is* single-threaded *if it contains at most one occurrence of an initial move.*

The next important definition is that of a strategy. Strategies determine unique responses for P (if any) and do not restrict O-moves.

Definition 6. *A* strategy *in an arena A (written as $\sigma : A$) is a non-empty prefix-closed subset of* single-threaded *plays in A such that:*

(i) whenever $sp_1, sp_2 \in \sigma$ and p_1, p_2 are P-moves then $p_1 = p_2$;
(ii) whenever $s \in \sigma$ and $so \in P_A$ for some O-move o then $so \in \sigma$.

We write $\mathsf{comp}\,(\sigma)$ *for the set of non-empty* complete plays *in σ, i.e. plays in which all questions have been answered.*

An IA_{f} term $\Gamma \vdash M : \theta$, where $\Gamma = x_1 : \theta_1, \cdots, x_n : \theta_n$, is interpreted by a strategy (denoted by $[\![\Gamma \vdash M : \theta]\!]$) in the arena $[\![\Gamma \vdash \theta]\!] = [\![\theta_1]\!] \times \cdots \times [\![\theta_n]\!] \Rightarrow [\![\theta]\!]$. The denotations are calculated compositionally starting from strategies corresponding to constants and free identifiers [1]. The latter are interpreted by identity strategies that copy moves across from one occurrence of the same arena to the other, subject to the constraint that the interactions must be plays. Strategies corresponding to constants are given in Figure 3, where the induced complete plays are listed. We use subscripts to indicate the origin of moves. Let $\sigma : A \Rightarrow B$ and $\tau : B \Rightarrow C$. In order to compose the strategies, one first defines an auxiliary set $\sigma^\dagger$ of (not necessarily single-threaded) plays on $A \Rightarrow B$ that are special interleavings of plays taken from σ (we refer the reader to [1] for details). Then $\sigma; \tau : A \Rightarrow C$ can be obtained by synchronising $\sigma^\dagger$ and τ on B-moves and erasing them after the synchronisation. The game-semantic interpretation captures contextual approximation as follows.

Theorem 1 ([1]). *$\Gamma \vdash M_1 \sqsubseteq\!\!\!\approx M_2$ if and only if* $\mathsf{comp}\,[\![\Gamma \vdash M_1]\!] \subseteq \mathsf{comp}\,[\![\Gamma \vdash M_2]\!]$.

Remark 1. $\sigma^\dagger$ is an interleaving of plays in σ that must itself be a play in $P_{A \Rightarrow B}$. For instance, only O is able to switch between different copies of σ and this can only happen

$$
\begin{array}{ll}
[\![\mathbf{skip}]\!] : [\![\mathsf{com}]\!] & run\ done \\
[\![i]\!] : [\![\mathsf{exp}]\!] & q\, i \\
[\![\mathbf{succ}]\!] : [\![\mathsf{exp}]\!]_1 \Rightarrow [\![\mathsf{exp}]\!] & q\, q_1\ \sum_{i=0}^{max} i_1\ ((i+1) \bmod max) \\
[\![\mathbf{pred}]\!] : [\![\mathsf{exp}]\!]_1 \Rightarrow [\![\mathsf{exp}]\!] & q\, q_1\ \sum_{i=0}^{max} i_1\ ((i-1) \bmod max) \\
[\![\mathbf{ifzero}_\beta]\!] : [\![\mathsf{exp}]\!]_3 \Rightarrow [\![\beta]\!]_2 \Rightarrow [\![\beta]\!]_1 \Rightarrow [\![\beta]\!] & \\
\qquad \sum_{q \vdash [\![\beta]\!]\, a} q\, q_3\, 0_3\, q_1\, a_1\, a + \sum_{q \vdash [\![\beta]\!]\, a} q\, q_3\, (\sum_{i=1}^{max} i_3)\, q_2\, a_2\, a & \\
[\![\mathbf{seq}_\beta]\!] : [\![\mathsf{com}]\!]_2 \Rightarrow [\![\beta]\!]_1 \Rightarrow [\![\beta]\!] & \sum_{q \vdash [\![\beta]\!]\, a} q\, run_2\, done_2\, q_1\, a_1\, a \\
[\![\mathbf{deref}]\!] : [\![\mathsf{var}]\!]_1 \Rightarrow [\![\mathsf{exp}]\!] & q\, read_1 \sum_{i=0}^{max} i_1\, i \\
[\![\mathbf{assign}]\!] : [\![\mathsf{var}]\!]_2 \Rightarrow [\![\mathsf{exp}]\!]_1 \Rightarrow [\![\mathsf{com}]\!] & run\ q_1\ \sum_{i=0}^{max} i_1\ write(i)_2\ ok_2\ done \\
[\![\mathbf{cell}_\beta]\!] : ([\![\mathsf{var}]\!]_{1,1} \Rightarrow [\![\beta]\!]_1) \Rightarrow [\![\beta]\!] & \\
\qquad \sum_{q \vdash [\![\beta]\!]\, a} q q_1 (read_{1,1}\, 0_{1,1})^* (\sum_{i=0}^{max} write(i)_{1,1}\ ok_{1,1} (read_{1,1}\, i_{1,1})^*)^* a_1 a & \\
[\![\mathbf{mkvar}]\!] : ([\![\mathsf{exp}]\!]_{2,1} \Rightarrow [\![\mathsf{com}]\!]_2) \Rightarrow [\![\mathsf{exp}]\!]_1 \Rightarrow [\![\mathsf{var}]\!] & \\
\qquad read\, q_1\, (\sum_{i=0}^{max} i_1\, i) + \sum_{i=0}^{max} write(i)\, run_2\, (q_{2,1}\, i_{2,1})^*\ done_2\ ok & \\
[\![\mathbf{while}]\!] : [\![\mathsf{exp}]\!]_2 \Rightarrow [\![\mathsf{com}]\!]_1 \Rightarrow [\![\mathsf{com}]\!] & run\, q_2\, (\sum_{i=1}^{max} i_2\, run_1\, done_1\, q_2)^*\, 0_2\ done
\end{array}
$$

Fig. 3. Strategies for constants. Only complete plays are specified.

after P plays in B. We shall discuss two important cases in detail, namely, $B = [\![\beta]\!]$ and $B = [\![\beta_k \to \cdots \to \beta_1 \to \beta]\!]$.

- If $B = [\![\beta]\!]$ then a new copy of σ can be started only after the previous one is completed. Thus $\sigma^\dagger$ in this case consists of iterated copies of σ.
- If $B = [\![\beta_k \to \cdots \to \beta_1 \to \beta]\!]$ then, in addition to the above scenario, a new copy of σ can be started by O each time P plays q_i (question from β_i). An old copy of σ can be revisited with a_i, which will then answer some unanswered occurrence of q_i. However, due to the bracketing condition, this will be possible only after all questions played after that q_i have been answered, i.e. when all copies of σ opened after q_i are completed. Thus, in this particular case, $\sigma^\dagger$ contains "stacked" copies of σ. Consequently, we can capture $X = \{\,\epsilon\,\} \cup \mathsf{comp}\,(\sigma^\dagger)$ in this case by equation

$$
\begin{array}{l}
X = \{\varepsilon\} \cup \bigcup \{\, q\, A_0\, q_{i_1}\, X\, a_{i_1}\, A_1 \ldots q_{i_m}\, X\, a_{i_m}\, A_m\, a\, X \mid \\
\qquad q A_0 q_{i_1} a_{i_1} A_1 \ldots q_{i_m} a_{i_m} A_m a \in \mathsf{comp}\,(\sigma)\}
\end{array}
$$

  where A_j's stand for (possibly empty and possibly different) segments of moves from A. Note that, in a play of σ, q_i will always be followed by a_i.

4 Upper bounds

We shall prove that contextual approximation of IA_3^2 terms can be decided in exponential space. Thanks to Lemma 1, this will imply that approximation of IA_3^k ($k \geq 2$) terms is in $(k-1)$-EXPSPACE. In Section 5 we will show that these bounds are tight.

This shows that by firing redexes of level higher than 3 we do not lose optimal complexity. However, if redexes of order 2 were also executed (i.e. first-order procedures were inlined), the problem would be reduced to IA_3^1 and the implied bound would

be 2-EXPTIME, which turns out suboptimal. In what follows, we show that contextual approximation of IA_3^2 terms is in EXPSPACE. To that end, we shall translate the terms to automata that represent their game semantics. The alphabet of the automata will consist of moves in the corresponding games. Recall that each occurrence of a base type in a type contributes distinct moves. In order to represent their origin faithfully, we introduce a labelling scheme based on subscripts.

First we discuss how to label occurrences of base types in types. Let Θ be a type of order at most 3. Then $\Theta \equiv \Theta_m \to \cdots \to \Theta_1 \to \beta$ and Θ_i's are of order at most 2. Consequently, for each $1 \leq i \leq m$, we have $\Theta_i \equiv \Theta_{i,m_i} \to \cdots \to \Theta_{i,1} \to \beta_i$ and $\Theta_{i,j}$'s are of order at most 1. Thus, we have $\Theta_{i,j} \equiv \beta_{i,j,m_{i,j}} \to \cdots \to \beta_{i,j,1} \to \beta_{i,j}$. Note that the above decomposition assigns a sequence of subscripts to each occurrence of a base type in Θ. Observe that $\mathsf{ord}(\Theta) = 3$ if and only if some occurrence of a base type gets subscripted with a triple. Next we are going to employ the subscripts to distinguish base types in IA_3 typing judgments.

Definition 7. *A* third-order typing template Ψ *is a sequence* $x_1 : \theta_1, \cdots, x_n : \theta_n, \theta$*, where* $\mathsf{ord}(\theta_i) \leq 2$ *(*$1 \leq i \leq n$*) and* $\mathsf{ord}(\theta) \leq 3$*.*

To label $\theta_1, \cdots, \theta_n, \theta$ we will use the same labelling scheme as discussed above but, to distinguish θ_i's from θ and from one another, we will additionally use superscripts x_i for the former. The labelling scheme will also be used to identify moves in the corresponding game. Recall that the game corresponding to a third-order typing template will have moves from $M_{[\![\theta_1]\!]} + \cdots + M_{[\![\theta_n]\!]} + M_{[\![\theta]\!]}$. The super- and subscripts will identify their origin in a unique way.

Example 1. Let $\Psi \equiv x_1 : (\mathsf{com} \to \mathsf{exp}) \to \mathsf{var}, x_2 : \mathsf{com} \to \mathsf{exp} \to \mathsf{var}, ((\mathsf{com} \to \mathsf{exp}) \to \mathsf{var}) \to \mathsf{com}$. Here is the labelling scheme for Ψ: $x_1 : (\mathsf{com}_{1,1}^{x_1} \to \mathsf{exp}_1^{x_1}) \to \mathsf{var}^{x_1}$, $x_2 : \mathsf{com}_2^{x_2} \to \mathsf{exp}_1^{x_2} \to \mathsf{var}^{x_2}$, $((\mathsf{com}_{1,1,1} \to \mathsf{exp}_{1,1}) \to \mathsf{var}_1) \to \mathsf{com}$. In the corresponding games, among others, we will thus have moves $run_{1,1}^{x_1}$, $run_2^{x_2}$, $q_1^{x_2}$, $read^{x_2}$, $run_{1,1,1}$ as well as run.

Our representation of game semantics will need to account for justification pointers. Due to the well-bracketing condition, pointers from answers need not be represented explicitly. Moreover, because of the visibility condition, in our case we only need to represent pointers from moves of the shapes $q_{i,j}^x$ and $q_{i,j,k}$. Such pointers must point at some moves of the form q_i^x and $q_{i,j}$ respectively. In order to represent a pointer we are going to place a hat symbol above both the source and target of the pointer, i.e. we shall also use "moves" of the form $\widehat{q_{i,j}^x}$, $\widehat{q_{i,j,k}}$ (sources) and $\widehat{q_i^x}$, $\widehat{q_{i,j}}$ (targets) - the latter hatted moves will only be used if the former exist in the sequence. Similarly to [10], we shall represent a single play by *several* sequences of (possibly hatted) moves under the following conditions:

- whenever a target-move of the kind discussed above is played, it may or may not be hatted in the representing sequences of moves,
- if a target-move is hatted, all source-moves pointing at the target move are also hatted,
- if a target-move is not hatted, no source-moves pointing at the move are hatted.

Note that this amounts to representing all pointers for a selection of possible targets, i.e. none, one or more (including all). Because the same $\widehat{\ }$-symbol is used to encode each pointer, in a single sequence there may still be ambiguities as to the real target of a pointer. However, among the representing plays we will also have plays representing pointers only to single targets, which suffice to recover pointer-related information. This scheme works correctly because only pointers from P-moves need to be represented and the strategies are deterministic (see the discussion at the end of Section 3 in [13]).

Example 2. The classic examples of terms that do need explicit pointers are the Kierstaad terms $\vdash K_1, K_2 : ((\mathsf{com}_{1,1,1} \to \mathsf{com}_{1,1}) \to \mathsf{com}_1) \to \mathsf{com}$ defined by $K_i \equiv \lambda f^{(\mathsf{com}\to\mathsf{com})\to\mathsf{com}}.f(\lambda x_1^{\mathsf{com}}.f(\lambda {x_2}^{\mathsf{com}}.x_i))$. To represent the corresponding strategies the following sequences of moves will be used (among others).

- K_1: $q\, q_1\, q_{1,1}\, q_1\, q_{1,1}\, q_{1,1,1}$ (zero targets), $q\, q_1\, \widehat{q_{1,1}}\, q_1\, q_{1,1}\, \widehat{q_{1,1,1}}$ (one target), $q\, q_1\, q_{1,1}\, q_1\, \widehat{q_{1,1}}\, q_{1,1,1}$ (one target), $q\, q_1\, \widehat{q_{1,1}}\, q_1\, \widehat{q_{1,1}}\, \widehat{q_{1,1,1}}$ (two targets).
- K_2: $q\, q_1\, q_{1,1}\, q_1\, q_{1,1}\, q_{1,1,1}$ (zero targets), $q\, q_1\, \widehat{q_{1,1}}\, q_1\, q_{1,1}\, q_{1,1,1}$ (one target), $q\, q_1\, q_{1,1}\, q_1\, \widehat{q_{1,1}}\, \widehat{q_{1,1,1}}$ (one target), $q\, q_1\, \widehat{q_{1,1}}\, q_1\, \widehat{q_{1,1}}\, \widehat{q_{1,1,1}}$ (two targets).

To represent strategies corresponding to IA_3^2-terms we are going to define an extension of visibly pushdown automata [4]. The alphabet will be divided push-, pop- and no-op-letters corresponding to possibly hatted moves. Additionally, we will use ϵ-transitions that can modify stack content, albeit using a distinguished stack alphabet.

Definition 8. *Let $\Psi = x_1 : \theta_1, \cdots, x_m : \theta_m, \theta$ be a third-order typing template and let $\mathcal{M} = M_{[\![\theta_1]\!]} + \cdots + M_{[\![\theta_n]\!]} + M_{[\![\theta]\!]}$. Below we shall refer to the various components of $\mathcal{M}$ using subscripts and superscripts according to the labelling scheme introduced earlier, also using q and a for questions and answers respectively. We define the sets $\Sigma_{\mathrm{push}}, \Sigma_{\mathrm{pop}}, \Sigma_{\mathrm{noop}}$ as follows.*

- $\Sigma_{\mathrm{push}} = \{q_{i,j,k}, \widehat{q_{i,j,k}} \mid q_{i,j,k} \in \mathcal{M}\} \cup \{q_{i,j}^{x_h}, \widehat{q_{i,j}^{x_h}} \mid q_{i,j}^{x_h} \in \mathcal{M}\}$
- $\Sigma_{\mathrm{pop}} = \{a_{i,j,k} \mid a_{i,j,k} \in \mathcal{M}\} \cup \{a_{i,j}^{x_h} \mid a_{i,j}^{x_h} \in \mathcal{M}\}$
- $\Sigma_{\mathrm{noop}} = (\mathcal{M} \setminus (\Sigma_{\mathrm{push}} \cup \Sigma_{\mathrm{pop}})) \cup \{\widehat{q_{i,j}} \mid q_{i,j,k} \in \mathcal{M}\} \cup \{\widehat{q_i^{x_h}} \mid q_{i,j}^{x_h} \in \mathcal{M}\}$

Σ_{push} and Σ_{pop} contain exclusively P- and O-moves respectively, while we can find both kinds of moves in Σ_{noop}. Let us write $\Sigma^O_{\mathrm{noop}}, \Sigma^P_{\mathrm{noop}}$ for subsets of Σ_{noop} consisting of O- and P-moves respectively. The states of our automata will be partitioned into states at which O is to move (O-states) and whose at which P should reply (P-states). Push-moves and ϵ-transitions are only available at P-states, while pop-transitions belong to O-states. No-op transitions may be available from any kind of state. Further, to reflect determinacy of strategies, P-states allow for at most one executable outgoing transition, which may be labelled with an element of Σ^P (push or no-op) or be silent (noop, push or pop).

Definition 9. *Let Ψ be a third-order typing template. A Ψ-automaton $\mathcal{A}$ is a tuple $(Q, \Sigma, \Upsilon, \delta, i, F)$ such that*

- *$Q = Q^O + Q^P$ is a finite set of states partitioned into O-states and P-states,*
- *$\Sigma = \Sigma^O + \Sigma^P$ is the finite transition alphabet obtained from Ψ as above, partitioned into O- and P-letters, where $\Sigma^O = \Sigma_{\mathrm{pop}} + \Sigma^O_{\mathrm{noop}}$ and $\Sigma^P = \Sigma_{\mathrm{push}} + \Sigma^P_{\mathrm{noop}}$,*

- $\Upsilon = \Upsilon^{\Sigma} + \Upsilon^{\epsilon}$ *is a finite stack alphabet partitioned into* Σ*-symbols and* ϵ*-symbols,*
- $\delta = \delta^O_{\mathrm{pop}} + \delta^O_{\mathrm{noop}} + \delta^P$ *is a transition function consisting of* $\delta^O_{\mathrm{pop}} : Q^O \times \Sigma_{\mathrm{pop}} \times \Upsilon_{\Sigma} \rightharpoonup Q^P$, $\delta^O_{\mathrm{noop}} : Q^O \times \Sigma^O_{\mathrm{noop}} \rightharpoonup Q^P$ *and* $\delta^P : Q^P \rightharpoonup (\Sigma_{\mathrm{push}} \times Q^O \times \Upsilon_{\Sigma}) + (\Sigma^P_{\mathrm{noop}} \times Q^O) + Q^P + (Q^P \times \Upsilon_{\epsilon}) + (\Upsilon_{\epsilon} \rightharpoonup Q^P)$,
- $i \in Q^O$ *is an initial state, and*
- $F \subseteq Q^O$ *is a set of final states.*

Ψ-automata are to be started in the initial state with empty stack. They will accept by final state, but whenever this happens the stack will be empty anyway. Clearly, they are deterministic. The set of words derived from runs will be referred to as the trace-set of $\mathcal{A}$, written $\mathcal{T}(\mathcal{A})$. We write $\mathcal{L}(\mathcal{A})$ for the subset of $\mathcal{T}(\mathcal{A})$ consisting of accepted words only. The Ψ-automata to be constructed will satisfy an additional run-time property called *P-liveness*: whenever the automaton reaches a configuration $(q, \gamma) \in Q^P \times \Upsilon$ from (i, ϵ), δ^P will provide a unique executable transition.

Remark 2. In what follows we shall reason about IA^2_3 terms by structural induction. The base cases are the constants and identifiers $\Gamma, f : \theta \vdash f : \theta$, where $\mathsf{ord}(\theta) \leq 2$. For inductive cases, we split the rule for application into linear application and contraction.

$$\frac{\Gamma \vdash M : \theta \to \theta' \quad \Delta \vdash N : \theta}{\Gamma, \Delta \vdash MN : \theta'}\ \mathsf{ord}(\theta \to \theta') \leq 2 \qquad \frac{\Gamma, x : \theta, y : \theta \vdash M : \theta'}{\Gamma, x : \theta \vdash M[x/y] : \theta'}$$

Note that the restriction on $\theta \to \theta'$ is consistent with the fact that the level of redexes cannot exceed 2 and free identifiers have types of order at most 2. The relevant λ-abstraction rule is

$$\frac{\Gamma, x : \theta \vdash M : \theta'}{\Gamma \vdash \lambda x^{\theta}.M : \theta \to \theta'}\ \mathsf{ord}(\theta \to \theta') \leq 3.$$

This stems from the fact that we are considering IA_3.

Lemma 2. *Let* $x_1 : \theta_1, \cdots, x_m : \theta_m \vdash M : \theta$ *be an* IA^2_3*-term and let* $\sigma = [\![x_1 : \theta_1, \cdots, x_m : \theta_m \vdash M : \theta]\!]$*. There exists a P-live* $(x_1 : \theta_1, \cdots, x_m : \theta_m, \theta)$*-automaton* $\mathcal{A}_M$*, constructible from* M *in exponential time, such that* $\mathcal{T}(\mathcal{A}_M)$ *and* $\mathcal{L}(\mathcal{A}_M)$ *represent respectively* σ *and* $\mathsf{comp}\,(\sigma)$ *(in the sense of our representation scheme).*

Proof. Translation by structural induction in IA^2_3. The base cases corresponding to the special constants can be resolved by constructing finite automata, following the description of the plays in Figure 3. For free identifiers, automata of a similar kind have already been constructed as part of the translation of normal forms in [13]. We revisit them below to show which moves must be marked to represent pointers.

Let θ be a second-order type. Then $x : \theta \vdash x : \theta$ is interpreted by the identity strategy, which has complete plays of the form $\sum_{q \vdash a} q q^x X a^x a$, where X is given by the context-free grammar below. When writing $\sum_{q \vdash a}$, we mean summing up over all pairs of moves of the indicated shape available in the associated arena $\mathcal{M}$ such that $q \vdash_{\mathcal{M}} a$. Below we also use the condition $\exists q.q_i \vdash q$ to exclude moves of the form q_i that do not enable any other questions (such moves are never targets of justification pointers).

$$X \quad \to \quad \epsilon \quad | \quad (\textstyle\sum_{q_i \vdash a_i} q^x_i q_i Y_i^* a_i a^x_i)\, X \quad | \quad (\sum_{\substack{q_i \vdash a_i \\ \exists q.q_i \vdash q}} \widehat{q^x_i} q_i (\widehat{Y_i})^* a_i a^x_i)\, X$$

$$Y_i \quad \to \quad \textstyle\sum_{q_{i,j} \vdash a_{i,j}} q_{i,j} q^x_{i,j} X a^x_{i,j} a_{i,j} \qquad \widehat{Y_i} \quad \to \quad \sum_{q_{i,j} \vdash a_{i,j}} q_{i,j} \widehat{q^x_{i,j}} X a^x_{i,j} a_{i,j}$$

To capture X, we can construct $\mathcal{A}_x$ as in [13], by pushing return addresses when reading $q^x_{i,j}, \widehat{q^x_{i,j}}$ and popping them at $a^x_{i,j}$. Note that this simply corresponds to interpreting recursion in the grammar.

λ-abstraction and contraction are interpreted by renamings of the alphabet, so it remains to consider the hardest case of (linear) application. The rule simply corresponds to composition: in any cartesian-closed category $[\![\Gamma, \Delta \vdash MN : \theta']\!]$ is equal (up to currying) to $[\![\Delta \vdash N : \theta]\!]; [\![\vdash \lambda x^\theta.\lambda\Gamma.Mx : \theta \to (\Gamma \to \theta')]\!]$. Note that in our case $\mathsf{ord}(\theta) \leq 1$, i.e. Remark 1 will apply and the strategy for MN can be obtained by running the strategy for M, which will call copies of N, whose interleavings will obey the stack discipline. To model the interaction, let us consider the moves on which the automata will synchronise. Since $\mathsf{ord}(\theta) \leq 1$, the moves that will interact will be of the form q, a, q_i, a_i from N's point of view and $q_k, a_k, q_{k,i}, a_{k,i}$ from M's viewpoint for some k. Thus, given $\mathcal{A}_M = (Q_M, \Sigma_M, \Upsilon_M, i_M, \delta_M, F_M)$ and $\mathcal{A}_N = (Q_N, \Sigma_N, \Upsilon_N, i_N, \delta_N, F_N)$, we let $\mathcal{A}_{MN} = (Q_{MN}, \Sigma_{MN}, \Upsilon_{MN}, i_M, \delta_{MN}, F_M)$, where

$$\begin{aligned} Q_{MN} &= Q_M + (Q^O_M \times Q_N) \\ \Sigma_{MN} &= (\Sigma_M \setminus \{q_k, a_k, q_{k,i}, a_{k,i}\}) + (\Sigma_N \setminus \{q_0, a_0, q_1, a_1\}) \\ \Upsilon^{\Sigma_{MN}}_{MN} &= \Upsilon_M + \Upsilon_N \\ \Upsilon^{\epsilon}_{MN} &= \Upsilon^{\epsilon}_M + \Upsilon^{\epsilon}_N + Q^O_M \end{aligned}$$

The decomposition of Σ_{MN} into push-, pop- and noop-letters is inherited from the constituent automata. We specify the transition function δ_{MN} below using derivation rules referring to transitions in $\mathcal{A}_M$ and $\mathcal{A}_N$. A push-transition reading x and pushing γ will be labelled with $\xrightarrow{x/\gamma}$. Dually, $\xrightarrow{x,\gamma}$ will represent a pop. $\widetilde{x}$ stands for any transition involving x, where x could also be ϵ.

– $\mathcal{A}_M$'s non-interacting transitions are copied over to $\mathcal{A}_{MN}$.

$$\frac{s \xrightarrow{\widetilde{x}}_{\mathcal{A}_M} s'}{s \xrightarrow{\widetilde{x}}_{\mathcal{A}_{MN}} s'} \quad x \in (\Sigma_M \setminus \{q_k, a_k, q_{k,i}, a_{k,i}\}) + \{\epsilon\}$$

– M calls N (left) and N returns from the call (right).

$$\frac{s \xrightarrow{q_k}_{\mathcal{A}_M} s' \qquad i_N \xrightarrow{q}_{\mathcal{A}_N} t}{s \xrightarrow{\epsilon}_{\mathcal{A}_{MN}} (s', t)} \qquad \frac{s' \xrightarrow{a_k}_{\mathcal{A}_M} s'' \qquad t \xrightarrow{a}_{\mathcal{A}_N} f \in F_N}{(s', t) \xrightarrow{\epsilon}_{\mathcal{A}_{MN}} s''}$$

– N's non-interacting transitions are copied over while keeping track of $\mathcal{A}_M$'s state.

$$\frac{t \xrightarrow{\widetilde{x}}_{\mathcal{A}_N} t'}{(s, t) \xrightarrow{\widetilde{x}}_{\mathcal{A}_{MN}} (s, t')} \qquad s \in Q^O_M,\ x \in (\Sigma_N \setminus \{q_0, a_0, q_1, a_1\}) \cup \{\epsilon\}$$

– N calls its argument (left) and the argument returns (right).

$$\frac{s \xrightarrow{q_{k,i}}_{\mathcal{A}_M} s' \qquad t \xrightarrow{q_i}_{\mathcal{A}_N} t'}{(s, t) \xrightarrow{\epsilon/t'}_{\mathcal{A}_{MN}} s'} \qquad \frac{s' \xrightarrow{a_{k,i}}_{\mathcal{A}_M} s'' \qquad t' \xrightarrow{a_i}_{\mathcal{A}_N} t''}{s' \xrightarrow{\epsilon,t'}_{\mathcal{A}_{MN}} (s'', t'')}$$

Note that the interaction involves moves that are not used to represent pointers, i.e. whenever pointers are represented they remain the same as they were in the original strategies, which is consistent with the definition of composition. The states in Q_{MN} are divided into O- and P-states as follows: $Q^O_{MN} = Q^O_M + (Q^O_M \times Q^O_N)$ and $Q^P_{MN} = Q^P_M + (Q^O_M \times Q^P_N)$. The correctness of the construction follows from the fact that it is a faithful implementation of legal interactions (see, e.g., [9]), as discussed in Remark 1. P-liveness follows from the fact the constituent strategies are P-live and that the construction simulates interaction sequences, including infinite chattering. □

Our next step will be to analyse the shape of reachable configurations of $\mathcal{A}_M$. We aim to understand how many elements of Υ_ϵ can occur consecutively on the stack.

Definition 10. *Suppose $(q, \gamma) \in Q \times (\Upsilon_\Sigma \cup \Upsilon_\epsilon)^*$. The ϵ-density of γ is defined to be the length of the longest segment in γ consisting solely of consecutive elements from Υ_ϵ.*

While the size of stacks corresponding to IA^2_3 terms is unbounded (consider, for instance, $x : \theta \vdash x : \theta$ with $\theta = (\mathsf{com} \to \mathsf{com}) \to \mathsf{com}$), ϵ-density turns out to be bounded. We shall prove that it is exponential with respect to the size of the original term. This will be crucial to obtaining our upper bound. The main obstacle to proving this fact is the case of composition MN. As discussed in Remark 1, M "stacks up" copies of N and we would first like to obtain a bound on the number of nested calls to N that are not separated by a move from Σ_{push} (such moves block the growth of ϵ-density). For this purpose, we go back to plays and analyse sequences in which the relevant questions are pending: a *pending* question is one that has been played but remains unanswered. Observe that sequences of pending questions are always alternating. We will not be interested in the specific questions but only in their kinds, as specified by the table below.

Question	q	q_i, q^x	$q_{i,j}, q^x_i$	$q_{i,j,k}, q^x_{i,j}$
Kind	0	1	2	3

Definition 11. *Let s be a play. We define $\mathsf{pend}(s)$ to be the sequence from $\{0,1,2,3\}^*$ obtained from s by restricting it to pending questions and replacing each question with the number corresponding to its kind.*

Thus, any non-empty even-length play s, $\mathsf{pend}(s)$ will match the expression $0(12 + 32)^*(1 + 3)$. We say that the (12)-*potential* of s is equal to k if k is the largest k such that $\mathsf{pend}(s) = \cdots (12)^k \cdots$. In other words, the (12)-potential of a play is the length of the longest segment $(12)^k$ in $\mathsf{pend}(s)$.

Lemma 3. *Let $\Gamma \vdash M : \theta$ be an IA^2_3-term. Then the (12)-potential of any play in $[\![\Gamma \vdash M]\!]$ is bounded and the bound b_M is exponential in the size of M.*

Lemma 3 is a key technical result needed to establish the following boundedness property that is satisfied by automata representing IA^2_3-terms.

Lemma 4. *Let $\Gamma \vdash M : \theta$ be an IA^2_3-term and consider $\mathcal{A}_M$ constructed in Lemma 2. There exists a bound d_M, exponential in the size of M, such that the ϵ-density of configurations reachable by $\mathcal{A}_M$ is bounded by d_M.*

Next we derive a bound on plays witnessing failure of contextual approximation in IA_3^2. Consider IA_3^2-terms $\Gamma \vdash M_1, M_2 : \theta$ and let $\sigma_i = [\![\Gamma \vdash M_i : \theta]\!]$ for $i = 1, 2$. Given a play, let its *height* be the maximum number of pending questions from Σ_{push} occurring in any of its prefixes. Note that, for plays from σ_i, this will be exactly the maximum number of symbols from Υ_Σ that will appear on the stack of $\mathcal{A}_{M_i}$ at any point of its computation.

Lemma 5. *There exists a polynomial p such that if $\mathsf{comp}\,\sigma_1 \setminus \mathsf{comp}\,\sigma_2$ is not empty then it contains a play of height $p(n_1 + n_2)$, where n_1, n_2 are the numbers of states in $\mathcal{A}_{M_1}$ and $\mathcal{A}_{M_2}$ respectively.*

Theorem 2. *For IA_3^2-terms $\Gamma \vdash M_1, M_2 : \theta$, one can decide $\Gamma \vdash M_1 \sqsubseteq\!\!\!\sim M_2$ in exponential space.*

Proof. Note that this boils down to testing emptiness of $\mathsf{comp}\,\sigma_1 \setminus \mathsf{comp}\,\sigma_2$. By Lemma 5, it suffices to guess a play whose height is polynomial in the size of $\mathcal{A}_{M_1}$, $\mathcal{A}_{M_2}$, i.e. exponential with respect to term size. Moreover, by Lemma 4, the ϵ-density of the corresponding configurations of $\mathcal{A}_{M_1}$ and $\mathcal{A}_{M_2}$ will also be exponential. Thus, in order to check whether a candidate s is accepted by $\mathcal{A}_{M_1}$ and rejected by $\mathcal{A}_{M_2}$, we will only need to consider stacks of exponential size wrt M_1, M_2. Consequently, the guess can be performed on the fly and verified in exponential space. Because NEXPSPACE=EXPSPACE, the result follows.

Corollary 1. *For $k \geq 2$, contextual approximation of IA_3^k-terms is in $(k-1)$-EXPSPACE.*

5 Lower bounds

Here we show that contextual approximation of IA_1^k-terms is $(k-1)$-EXPSPACE-hard for $k \geq 2$. Note that this matches the upper bound shown for IA_3^k-terms and will allow us to conclude that contextual approximation in IA_1^k, IA_2^k and IA_3^k is $(k-1)$-EXPSPACE-complete. Our hardness results will rely on nesting of function calls and iteration afforded by higher-order types. Below we introduce the special types and terms to be used.

Let $k, n \in \mathbb{N}$. Define the type $\overline{n}$ by $\overline{0} = \mathsf{com}$ and $\overline{n+1} = \overline{n} \to \overline{n}$. Note that $\mathsf{ord}(\overline{n}) = n$. Also, let $\mathsf{Exp}(k, n)$ be defined by $\mathsf{Exp}(0, n) = n$ and $\mathsf{Exp}(k+1, n) = 2^{\mathsf{Exp}(k,n)}$. Given $k \geq 2$, consider the term $\mathsf{twice}_k = \lambda x^{\overline{k-1}}.\lambda y^{\overline{k-2}}.x(xy) : \overline{k}$.

Definition 12. *Let $k \geq 2$. Writing $M^n N$ as shorthand for $\underbrace{M(M \cdots (M}_{n} N) \cdots)$, let us define a family of terms $\{\mathsf{nest}_{n,k}\}$ with $f : \overline{1}, x : \overline{0} \vdash \mathsf{nest}_{n,k} : \overline{0}$ by taking $\mathsf{nest}_{n,k} \equiv (\mathsf{twice}_k^n\ g_{k-1}) g_{k-2} \cdots g_1 g_0$, where $g_0 \equiv x$, $g_1 \equiv f$ and $g_i \equiv \mathsf{twice}_i$ for $i > 1$.*

The terms have several desirable properties, summarised below.

Lemma 6. *Let $k \geq 2$. $\mathsf{nest}_{n,k}$ belongs to IA_2^k, has polynomial size in n and is β-reducible to $f^{\mathsf{Exp}(k-1,n)} x$.*

Note that the nested applications of f in $f^{\mathsf{Exp}(k-1,n)} x$ are akin to generating a stack of height $\mathsf{Exp}(k-1, n)$. We shall exploit this in our encodings. Note that, by substituting $\lambda c^{\mathsf{com}}.c; c$ for f in $f^{\mathsf{Exp}(k-1,n)} x$, we obtain a term that *iterates* x as many as $\mathsf{Exp}(k, n)$ times, i.e. $\mathsf{Exp}(k-1, n)$-fold nesting is used to simulate $\mathsf{Exp}(k, n)$-fold iteration.

Simulating Turing machines

Let w be an input word. Let $n = |w|$, $l = \mathsf{Exp}(k-1, n)$ and $N = \mathsf{Exp}(k, n)$. We shall consider a deterministic Turing machine T running in $SPACE(l)$ and $TIME(N)$ and simulate T's behaviour on w. This suffices to establish $SPACE(l)$-hardness.

We start off with the description of an encoding scheme for configurations of T. We shall represent them as strings of length l over an alphabet Σ_T, to be specified later. We shall write Config_T for the subset of $(\Sigma_T)^l$ corresponding to configurations. The encoding of the initial configuration will be denoted by c_{init} and we shall write Accept_T for the set of representations of accepting configurations. Given $c \in \mathsf{Config}_T$, we write $\mathsf{next}(c)$ for the representation of the successor of c according to T's transition function. Let us introduce a number of auxiliary languages that will play an important role in the simulation. We write c^R for the reverse of c.

Definition 13. *Let $\Sigma_T^{\#} = \Sigma_T + \{\#\}$. We define the languages $\mathcal{L}_0, \mathcal{L}_1 \subseteq (\Sigma_T)^*$ and $\mathcal{L}_2, \mathcal{L}_3, \mathcal{L}_4 \subseteq (\Sigma_T^{\#})^*$ as follows.*

$$\mathcal{L}_0 = \{c_{init}\} \qquad \mathcal{L}_1 = \mathsf{Accept}_T \qquad \mathcal{L}_2 = \{c^R \# \mathsf{next}(c) \mid c \in \mathsf{Config}_T\}$$
$$\mathcal{L}_3 = \{c \# \mathsf{next}(c)^R \mid c \in \mathsf{Config}_T\} \qquad \mathcal{L}_4 = \{c \# d^R \mid c \in \mathsf{Config}_T, d \neq \mathsf{next}(c)\}$$

Lemma 7. *There exists a representation scheme for configurations of T such that Σ_T is polynomial in the size of T, w and the following properties hold.*

1. *There exist deterministic finite-state automata $\mathcal{A}_0, \mathcal{A}_1$, constructible from T, w in polynomial time, such that $L(\mathcal{A}_0) \cap (\Sigma_T)^l = \mathcal{L}_0$ and $L(\mathcal{A}_1) \cap (\Sigma_T)^l = \mathcal{L}_1$.*
2. *For any $i = 2, 3, 4$, there exists a deterministic pushdown automaton $\mathcal{A}_i$, constructible from T, w in polynomial time, such that $L(\mathcal{A}_i) \cap ((\Sigma_T)^l \#(\Sigma_T)^l) = \mathcal{L}_i$. Moreover, transitions of the automata are given by three transition functions $\delta_{\mathrm{push}} : Q^{\mathrm{push}} \times \Sigma_T \to Q^{\mathrm{push}} \times \Upsilon$, $\delta_{\mathrm{noop}} : Q^{\mathrm{push}} \times \{\#\} \to Q^{\mathrm{pop}}$ and $\delta_{\mathrm{pop}} : Q^{\mathrm{pop}} \times \Sigma_T \times \Upsilon \to Q^{\mathrm{pop}}$, the initial state belongs to Q^{push} and the automaton accepts by final state. I.e., the automata will process elements of $(\Sigma_T)^l \#(\Sigma_T)^l$ by performing push-moves first, then a noop move for $\#$ and, finally, pop-moves.*

Remark 3. Note that in the above lemma we had to use intersection with $(\Sigma_T)^l$ (resp. $(\Sigma_T)^l \#(\Sigma_T)^l$) to state the correctness conditions with respect to Config_T, because the automata will not be able to count up to l. However, in our argument, we are going to use the nesting power of IA_1^k to run their transition functions for suitably many steps (l and $2l+1$ respectively).

The significance of the languages $\mathcal{L}_0, \mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3, \mathcal{L}_4$ stems from the fact that they are building blocks of two other languages, $\mathcal{L}_5$ and $\mathcal{L}_6$, which are closely related to the acceptance of w by T.

Lemma 8. *Consider the languages $\mathcal{L}_5, \mathcal{L}_6 \subseteq (\Sigma_T^{\#})^*$ defined by $\mathcal{L}_5 = \{c_{init} \# c_1^R \# d_1 \# \cdots c_N^R \# d_N \# f^R \mid c_j \in \mathsf{Config}_T, f \in \mathsf{Accept}_T, \forall_i \mathsf{next}(c_i) = d_i\}$ and $\mathcal{L}_6 = \{c_1 \# d_1^R \# \cdots c_N \# d_N^R \mid c_j \in \mathsf{Config}_T, \exists_i \mathsf{next}(c_i) \neq d_i\}$. Then T accepts w if and only if $\mathcal{L}_5 \not\subseteq \mathcal{L}_6$.*

Proof. Note that if T accepts w then the sequence of (representations of the) configurations belonging to the accepting run, in which every other representation is reversed, gives rise to a word that belongs to $\mathcal{L}_5$ but not to $\mathcal{L}_6$.

Conversely, if a word $c_{init} \# c_1^R \# d_1 \# \cdots c_N^R \# d_N \# f^R \in \mathcal{L}_5$ does not belong to $\mathcal{L}_6$ then $c_1 = \mathsf{next}(c_{init})$, $c_{i+1} = \mathsf{next}(d_i)$ $(i = 1, \cdots, N-1)$ and $f = \mathsf{next}(d_N)$. Thus, the word actually represents an accepting run on w. □

Our hardness argument consists in translating the above lemma inside IA_1^k. To that end, we shall show how to capture $\mathcal{L}_2, \mathcal{L}_3, \mathcal{L}_4$ and, ultimately, $\mathcal{L}_5$ and $\mathcal{L}_6$, using IA_1^k terms. We shall work under the assumption that $\Sigma_T^\# = \{0, \cdots, max\}$. Note, though, that the results can be adapted to any $max > 0$ by encoding $\Sigma_T^\#$ as sequences of exp-values. Similarly, using multiple exp-valued variables, IA-terms can store values that are bigger than max. We shall take advantage of such storage implicitly (e.g. for state values or stack elements), but the number of extra variables needed for this purpose will remain polynomial.

Definition 14. *We shall say that an* IA*-term* $z : \mathsf{exp} \vdash M : \mathsf{com}$ *captures* $L \subseteq (\Sigma_T^\#)^*$ *if* $\mathsf{comp}\,([\![z \vdash M]\!]) = \{run\, q^z\, (a_1)^z\, q^z\, (a_2)^z \cdots q^z\, (a_k)^z\, done \mid a_1 a_2 \cdots a_k \in L\}$.

Example 3. The term $z : \mathsf{exp} \vdash M_\# : \mathsf{com}$, where $M_\# \equiv$ if $z = \#$ then $\mathbf{skip}$ else Ω, captures $\{\#\}$. In our constructions we often write [condition] to stand in for the assertion if (condition) then $\mathbf{skip}$ else Ω.

Lemma 9. *There exist* IA_1^k*-terms* $z : \mathsf{exp} \vdash M_0, M_1 : \mathsf{com}$*, constructible from* T, w *in polynomial time, capturing* $\mathcal{L}_0, \mathcal{L}_1$ *respectively.*

Lemma 10. *There exists an* IA_1^k*-term* $z : \mathsf{exp} \vdash M_2 : \mathsf{com}$*, constructible from* T, w *in polynomial time, which captures* $\mathcal{L}_2$*.*

Thanks to the last two lemmas we are now ready to capture $\mathcal{L}_5$.

Lemma 11. *There exists an* IA_1^k*-term* $z : \mathsf{exp} \vdash M_5 : \mathsf{com}$*, constructible in polynomial time from* T, w*, which captures* $\mathcal{L}_5$*.*

Proof. Note that a word from $\mathcal{L}_5$ contains $N = \mathsf{Exp}(k, n)$ segments from $\mathcal{L}_2$. To account for that, it suffices to use N copies of $M_\#; M_2$. However, for a polynomial-time reduction, we need to do that succinctly. Recall that $\mathsf{nest}_{n,k}$ gives us l-fold nesting of functions, where $l = \mathsf{Exp}(k-1, n)$. Consequently, N-fold iteration can be achieved by l-fold nesting of $\lambda c^{\mathsf{com}}.c; c$. Thus, we can take

$$M_5 \equiv M_0; \mathsf{nest}_{n,k}[\lambda c^{\mathsf{com}}.c; c/f, (M_\#; M_2)/x]; M_\#; M_1.$$

To complete the hardness argument (by restating Lemma 8 using IA_1^k terms), we also need to capture $\mathcal{L}_6$. Because of the existential clause in its definition we need to use a slightly different capture scheme.

Lemma 12. *There exists an* IA_1^k*-term* $z : \mathsf{exp}, FLAG : \mathsf{var} \vdash M_6' : \mathsf{com}$*, constructible in polynomial time from* T, w*, such that* $\mathsf{comp}\,([\![z, FLAG \vdash M_6']\!]) = \{run\, q^z\, (a_1)^z\, q^z\, (a_2)^z \cdots q^z\, (a_k)^z\, done \mid a_1 a_2 \cdots a_k \in \mathcal{L}_3\} \cup \{run\, q^z\, (a_1)^z\, q^z\, (a_2)^z \cdots q^z\, (a_k)^z\, write(1)^{FLAG}\, ok^{FLAG}\, done \mid a_1 a_2 \cdots a_k \in \mathcal{L}_4\}$.

Lemma 13. *There exists an* IA_1^k*-term* $z : \mathsf{exp} \vdash M_6 : \mathsf{com}$*, constructible in polynomial time from* T, w*, which captures* $\mathcal{L}_6$*.*

Proof. It suffices to run M_6' for $N+1$ steps and check whether the flag was set:

$$M_6 \equiv \textbf{new}\ FLAG\ \textbf{in}\ (FLAG := 0; \mathsf{nest}_{n,k}[\lambda c^{\mathsf{com}}.c; c/f, (M_6'; M_\#)/x]; M_6'; [!FLAG = 1])$$

Theorem 3. *Contextual approximation between* IA_1^k *terms is* $(k-1)$*-EXPSPACE-hard.*

Proof. By Lemmas 8, 11, 13, for any Turing machine T running in $SPACE(\mathsf{Exp}(k-1, n))$ and $TIME(\mathsf{Exp}(k, n))$ and an input word w, there exist IA_1^k-terms $x : \mathsf{exp} \vdash M_5, M_6$, constructible from T, w in polynomial time, such that T accepts w if and only if M_5 does not approximate M_6. This implies $(k-1)$-EXPSPACE-hardness. □

6 Conclusion

We have shown that contextual approximation in $\mathsf{IA}_1^k, \mathsf{IA}_2^k, \mathsf{IA}_3^k$ is $(k-1)$-EXPSPACE-complete. The algorithm that leads to these optimal bounds reduces terms to IA_3^2 (with possibly $(k-2)$-fold exponential blow-up) and we use a dedicated EXPSPACE procedure for IA_3^2 exploiting game semantics and decision procedures for a special kind of pushdown automata. In particular, the results show that untamed β-reduction would yield suboptimal bounds, but selective β-reduction of redexes up to level 3 does not jeopardise complexity. The bounds above apply to open higher-order terms, i.e. IA_i $(i > 0)$ terms, for which the problem of contextual approximation is difficult to attack due to universal quantification over contexts.

Our work also implies bounds for contextual approximation of IA_0^k terms, i.e. closed terms of base type. Conceptually, this case is much easier, because it boils down to testing termination. In this case our techniques can be employed to obtain better upper bounds for IA_0^k than those for IA_1^k ($(k-1)$-EXPSPACE). For a start, like for IA_1^k, we can reduce IA_0^k terms (at $(k-2)$-fold exponential cost) to IA_0^2. Then termination in IA_0^2 can be checked in exponential *time* by constructing pushdown automata via Lemma 2 and testing them for emptiness (rather than inclusion). Since emptiness testing of pushdown automata can be performed in polynomial time and the automata construction in Lemma 2 costs a single exponential, this yields an EXPTIME upper bound for termination in IA_0^2. Consequently, termination in IA_0^k $(k \geq 2)$ can be placed in $(k-1)$-EXPTIME, though it is not clear to us whether this bound is optimal. For completeness, let us just mention that termination in IA_0^0 and IA_0^1 is PSPACE-complete due to presence of variables and looping (membership follows from the corresponding upper bounds for contextual equivalence).

Another avenue for future work is $\mathsf{IA}_1^k, \mathsf{IA}_2^k, \mathsf{IA}_3^k$ contextual equivalence. Of course, our upper bounds for approximation also apply to contextual equivalence, which amounts to two approximation checks. However, one might expect better bounds in this case given that our hardness argument leans heavily on testing inclusion.

Finally, one should investigate how our results can be adapted to the call-by-value setting. An educated guess would be that, in the analogous fragment of ML, the reduction of redexes up to order 3 (rather than 2) should be suppressed in order to obtain accurate complexity estimates.

References

1. S. Abramsky and G. McCusker. Linearity, sharing and state: a fully abstract game semantics for Idealized Algol with active expressions. In P. W. O'Hearn and R. D. Tennent, editors, *Algol-like languages*, pages 297–329. Birkhaüser, 1997.
2. S. Abramsky and G. McCusker. Game semantics. In H. Schwichtenberg and U. Berger, editors, *Logic and Computation*. Springer-Verlag, 1998. Proceedings of the 1997 Marktoberdorf Summer School.
3. A. Ahmed, D. Dreyer, and A. Rossberg. State-dependent representation independence. In *Proceedings of POPL*, pages 340–353. ACM, 2009.
4. R. Alur and P. Madhusudan. Visibly pushdown languages. In *Proceedings of STOC'04*, pages 202–211, 2004.
5. A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *Proceedings of CONCUR*, volume 1243 of *Lecture Notes in Computer Science*, pages 135–150. Springer, 1997.
6. R. Colvin, I.J. Hayes, and P.A. Strooper. Calculating modules in contextual logic program refinement. *Theory and Practice of Logic Programming*, 8(01):1–31, 2008.
7. S. Fortune, D. Leivant, and M. O'Donnell. The expressiveness of simple and second-order type structure. *J. ACM*, 30:151–185, 1983.
8. D. R. Ghica and G. McCusker. Reasoning about Idealized Algol using regular expressions. In *Proceedings of ICALP*, volume 1853 of *Lecture Notes in Computer Science*, pages 103–115. Springer-Verlag, 2000.
9. R. Harmer. *Games and full abstraction for non-deterministic languages*. PhD thesis, University of London, 2000.
10. D. Hopkins, A. S. Murawski, and C.-H. L. Ong. A fragment of ML decidable by visibly pushdown automata. In *Proceedings of ICALP*, volume 6756 of *Lecture Notes in Computer Science*, pages 149–161. Springer, 2011.
11. A. S. Murawski. On program equivalence in languages with ground-type references. In *Proceedings of IEEE Symposium on Logic in Computer Science*, pages 108–117. Computer Society Press, 2003.
12. A. S. Murawski. Games for complexity of second-order call-by-name programs. *Theoretical Computer Science*, 343(1/2):207–236, 2005.
13. A. S. Murawski and I. Walukiewicz. Third-order Idealized Algol with iteration is decidable. In *Proceedings of FOSSACS*, volume 3441 of *Lecture Notes in Computer Science*, pages 202–218. Springer, 2005.
14. A. M. Pitts. Operational semantics and program equivalence. In *Proceedings of APPSEM*, volume 2395 of *Lecture Notes in Computer Science*, pages 378–412. Springer, 2000.
15. J. C. Reynolds. The essence of Algol. In J. W. de Bakker and J.C. van Vliet, editors, *Algorithmic Languages*, pages 345–372. North Holland, 1978.
16. M. Sipser. *Introduction to the Theory of Computation*. PWS Publishing Company, 1997.
17. A. Turon, D. Dreyer, and L. Birkedal. Unifying refinement and hoare-style reasoning in a logic for higher-order concurrency. In *Proceedings of ICFP'13*, pages 377–390, 2013.